Bachelor Final Project

Bachelor's Degree in Industrial Technologies Engineering (GETI)

Optimization of the communications system and addition of new functionalities to the line tracking system for a set of robots

REPORT

June 24, 2019

Author: Marc Salvadó Benasco

Supervisor: Arnau Dòria Cerezo

Call: June 2019

Barcelona School of
Industrial Engineering

Abstract

The aims of this project are the implementation of an effective communications system of a set of line tracking robots and the addition of new functionalities to their line tracking system.

This project starts from a system composed of a set of robots that successfully track lines and send information to a PC. Even though there is already a communications system that receives and decodes these information, it doesn't offer a functional user interface that effectively allows the user to analyse the received information and to send commands to the robots in a flexible and dynamic manner.

Both communications and line tracking systems are a result of previous works that went improving the system performance and adding new functionalities to it. This project then hopes to solve the need of an effective way to analyse the data sent from the robots and to add new functionalities to the line tracking system, such as the option of changing lane or the recovery from a loss.

Contents

List of Figures

1 Glossary

ARM: Family of reduced instruction set computing (RISC) architectures for computer processors, configured for various environments. Widely used in embedded systems because of its low price and low energetic consume.

Microcontroller: Small computer on a single integrated circuit specialised in controlling electronic kits, which includes a CPU, a small quantity of memory and I/O (input/output) units.

CPU (Central Processing Unit): Electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logic, controlling, and input/output (I/O) operations specified by the instructions.

Embedded system: Computer system designed to execute a few specific functions, often in real time, which is totally encapsulated by the device that it controls.

C: General-purpose, imperative computer programming language supporting structured programming, lexical variable scope, and recursion, while a static type system prevents unintended operations.

Function (programming): A function is a unit of code that is often defined by its role within a greater code structure. Specifically, a function contains a unit of code that works on various inputs, many of which are variables, and produces concrete results involving changes to variable values or actual operations based on the inputs.

GUI: A Graphical User Interface is a form of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation.

Robot ID: In the IOC laboratory, each robot has an ID referring the last three numbers of its IP address. For example, the robot with the IP address *192.168.173.100*, its ID is *100*.

Data package: In this project, we will refer a data packages as a package sent by each robot and processed by the server, and contains information about the robots local variables.

Order package: An order package will be referred as a packages sent to a robot or a group of robots from the user interface.

Initialisation package: An initialisation package is referred as a package that tells the server how to interpret the following state packages.

State package: A state package is referred as a package that is sent regularly every T_s after the robot initialisation process and it contains the numeric values of its local variables.

T_s: Time interval of a robot package delivery.

PCB: Printed Circuit Board.

LiPo battery: Lithium-ion polymer battery, a rechargeable battery of lithium-ion technology using a polymer electrolyte instead of a liquid electrolyte.

2 Introduction

This project has been developed at IOC-UPC (Institute of Industrial and Control Engineering) and is the continuation of previous projects consisting in the assemblage and the programming of line tracking robots, the creation of a communications system via Wi-Fi that allows the exchange of information between the robots and a PC, and the improvement of the line tracking control system.

At the start of this project, the robots are already able to send packages with information about their local variables, but the user interface doesn't allow to select which of these variables are to be plotted, but they are all fit in a small scope that doesn't offer a good analysis for the user. Besides, the way it decodes the received data only works if all the robots are sending the same variables and in the same order. In addition, all the orders have to be typed and sent individually to each robot, which makes coordinating them more difficult. Finally, the user interface is run locally in a Python GUI, which doesn't allow other devices to access it.

2.1 Aims of the project

The first aim the of this project is to create a useful communications system that replaces the current user interface and the way the received data is handled. Firstly, the new user interface must enable several devices that are connected to the same wireless network to access it, and at the same time, it must be easy to use and provide a high level of customisation. And secondly, a new system to handle the received data must be created in order to allow differently programmed robots to send data simultaneously, without the different format of their data causing trouble.

The second aim of this project regards the robots line tracking system and consists in the addition of new functionalities such as lane changing or recovering from a loss. The lane change performance will be added as a *function* in the robots script, that will be activated when receiving an order from the PC in the same way orders such as *start* and *stop* are already processed. Conversely, the recovery from loss functionality will interfere directly to the way the robots calculate their trajectory and will provide robustness to the model in both usual line tracking and lane change situations.

2.2 Scope of the project

The analysis of the robots variables is key when applying any tracking model or new functionality to the robots. The feedback allows us to check whether the model has any theoretical mistake or whether any of the robots sensors aren't working, as well as to calculate the real value of constants, such as the friction coefficient or the moment of inertia. The new communications system not only solves this problem, but also allows any device to access the user interface via Wi-Fi, which opens up the possibility of operating remotely from another office or even from home.

On the other hand, the lane change functionality extends the robots trajectories. Potentially, this can be applied autonomously when the robot finds an obstacle on the current lane, or in coordination with other robots when they must handle overtakings.

3 Description of the components

The line tracking robots are basically composed of a chassis and two driving wheels with their respective DC engines, plus a support multi-directional roller ball at the back of the robot; a *STM32F4 Discovery* development board and its PCB; a *POLOLU-960* optic sensors board; an *AeroEnergy* LiPo battery; 3 *DFROBOT-SEN0001* ultrasound sensors, whose calibration is currently being carried out by another student; and an *ESP8266* Wi-Fi module that will be in charge of the communication between the robot and the PC.

In addition, a wireless network is required, as well as a desktop computer that manages the communications between the robot and the other devices connected to the network.

Some of these components will be described below. The choice and the analysis of these components are not among the goals of this project, but were already predetermined.

3.1 The robots and their components

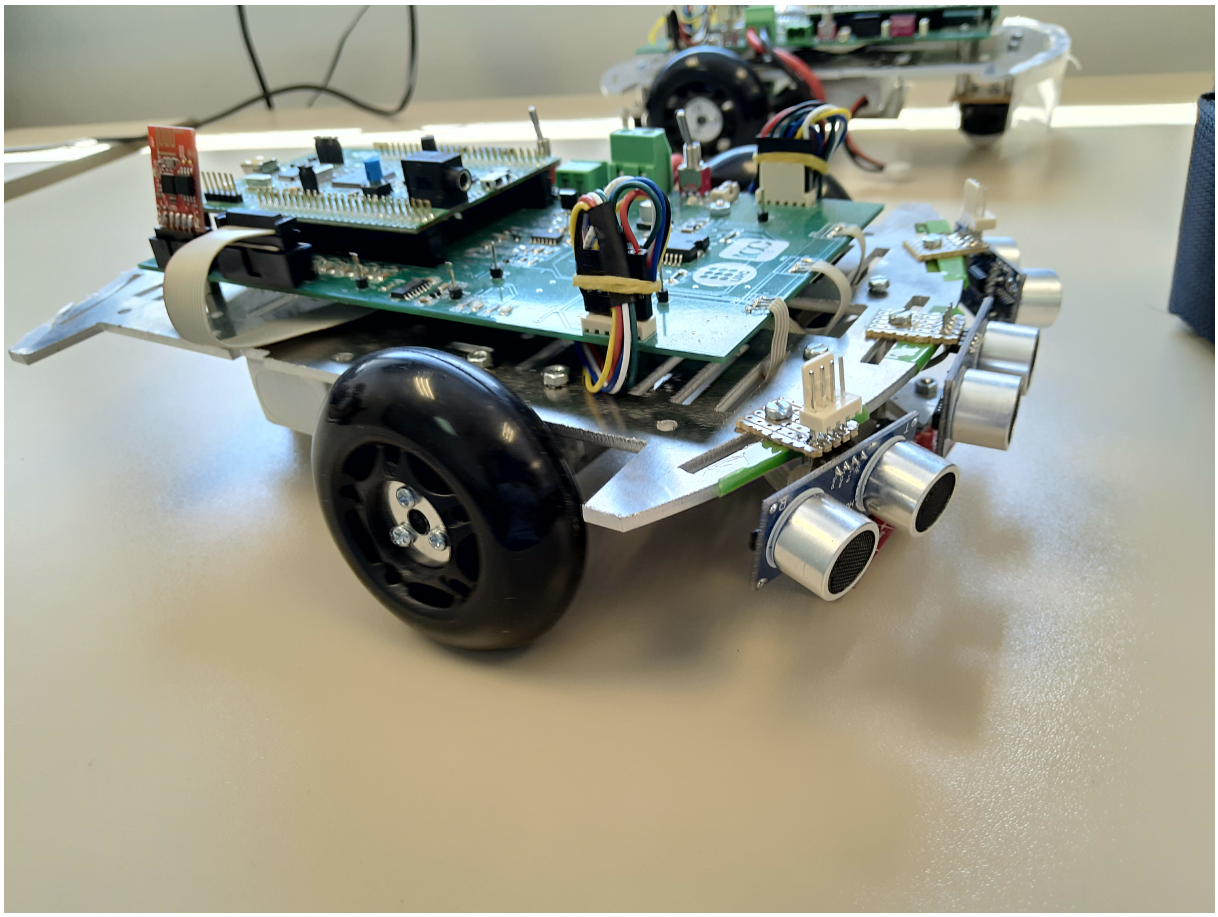

Figure 1: Picture of a robot used in the experiments of this project

3.1.1 STM32F4 microcontroller

The STM32F4DISCOVERY Discovery kit is a development board that allows users to easily develop applications with the STM32F407VG high performance microcontroller with the ARM® Cortex®-M4 32-bit core. It includes everything required either for beginners or for experienced users to get quickly started. [4] This development board processes the data received from the sensors, calculates a proper response and applies to corresponding current to the engines; and at the same time, with a lower priority, sends packages to the server and receives orders from it.

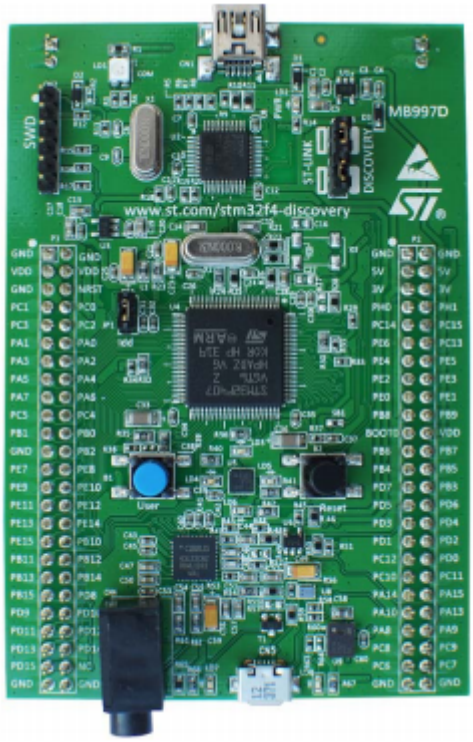

Figure 2: Picture of a STM32F4 Discovery development board. Source: [4]

3.1.2 ESP8266 Wi-Fi module

The ESP8266 is a low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability produced by manufacturer Espressif Systems in Shanghai, China. [5]
It comprises a CPU, RAM memory, a Wi-Fi aerial and a serial port. [6]

According to the conclusions of an analysis in a previous project [1], which tested and compared the behaviour of this Wi-Fi module, the package transmission rate must be 25 packages per second for an optimal performance.

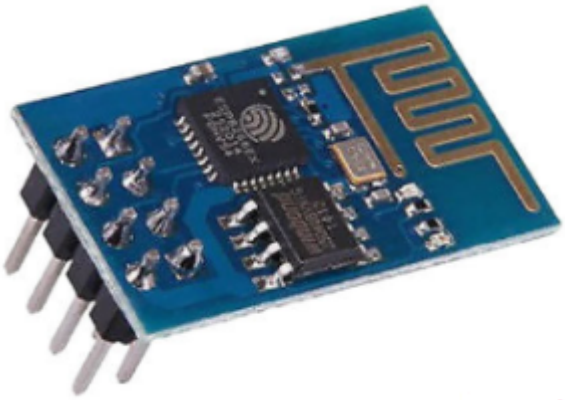

Figure 3: Picture of an ESP8266 Wi-Fi module. Source: [7]

3.1.3 POLOLU-960 optic sensors board

The POLOLU-960 optic sensors board contains 8 optic sensors whose analogue outputs are used to measure the distance between the robot and the tracking line. The board has a size of 75x12,7 [mm]. [10]

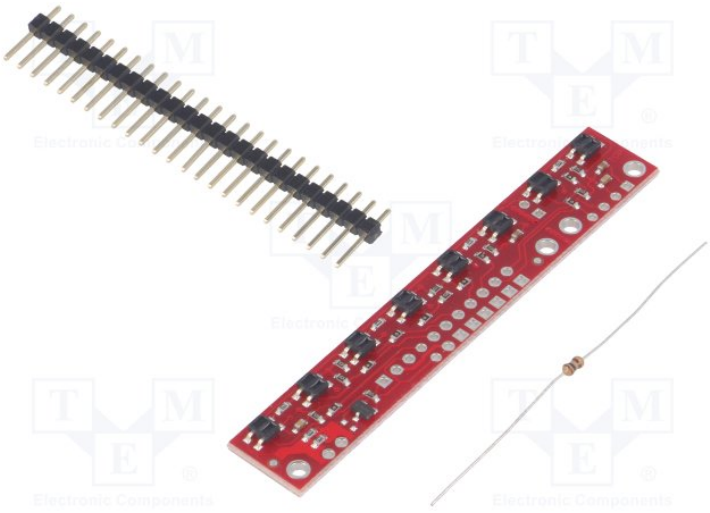

Figure 4: Picture of a POLOLU-960 optic sensors board. Source: [10]

3.2 Internet Communication Protocol (IP)

A communication protocol is a set of rules and technical specifications that allow communication between ends reliably. The IP Protocol requires that there is a number, called the IP address, that unequivocally identifies each device connected to Internet [8]. IP has then the task of delivering packets, called *datagrams*, from the source host to the destination host solely based on these IP addresses in the package headers. The device that manages and delivers these datagrams to

the corresponding address is called a *router*.

There are two main types of transport mechanisms: *User Datagram Protocol (UDP)* and *Transmission Control Protocol (TCP)*. The main difference between them is that TCP guarantees a reliable data transfer and takes a heavy verification process, while UDP is much simpler and focuses on the transmission speed, which makes it more suitable for this project than TCP. [9]

To conclude the previous information, the transport protocol used in the communication process between the robots and the server is UDP.

3.3 Desktop computer

A desktop computer is required to run the Python script that will handle the communication process: the packages collection, decryption and delivery to the user interface; as well as the orders delivery to the robots. This computer will be referred as the *server* of all the devices that are connected to the same network, the *clients*, which will be able to access the user interface and to send orders to the robots.

3.4 Wi-Fi access point and router

A Wi-Fi access point, which creates the network, and a router, which redirects the packages to their corresponding recipients, are required to allow the server to communicate with the robots by sending and receiving packages from them. Usually, the Wi-Fi access point comes along with the router.

4 Communications system

The communications system comprises a set of elements that enable the exchange of information between the robots and the server. Even though the ESP8266 Wi-Fi module could allow the robots to send packages to each other if so they were programmed, at the moment all the packages sent by the robots must go directly to the server.

The server, which is the PC with the IP address where all packages are redirected, processes both data packages and order packages. The programme in charge of this process is written in Python and leans on TXT and CSV documents to store the packages. The user interface, however, comprises a set of PHP documents.

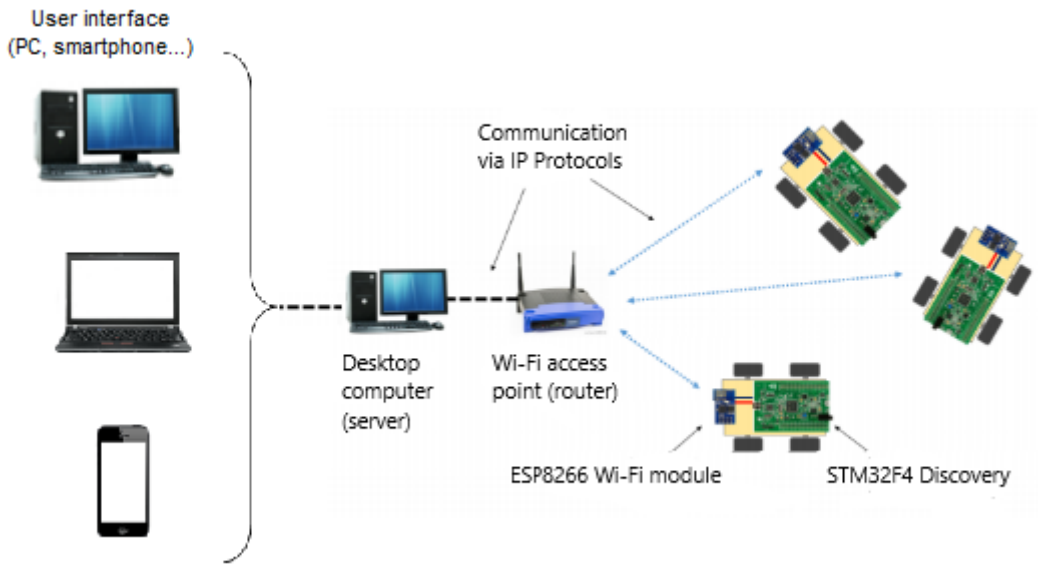

Figure 5: General scheme of the communications system. This is an extension of a picture from [1].

As shown in Figure 5, the communication process has two ends: the user interface and the robots. To connect them, there must be a Wi-Fi access point that receives the packages sent by the robots and redirects them to the server, which will store this data in some files that will be accessed by the user interface. At the same time, the user will tell the user interface to send an order, which will be properly coded by the server and sent by the router to the corresponding robots.

In Section 4.1 is an explanation of the general functioning of the communications system, whereas in Section 4.2 is an detailed description of how the received data is stored in files and simultaneously accessed by the user interface.

From now on, when referring the communication between the server and the robots, the Wi-Fi access point will be omitted in order to ease the future explanations, as its use has been clearly explained and will not provide more useful information.

4.1 Structure of the communications system

The communications system handles two flows of packages: the ones that are sent by the robots to the user interface, which contain information about the robots local variables and that here will be called *data packages*; and the ones that are sent by the user interface, which contain orders for the robots and that will be called *order packages*.

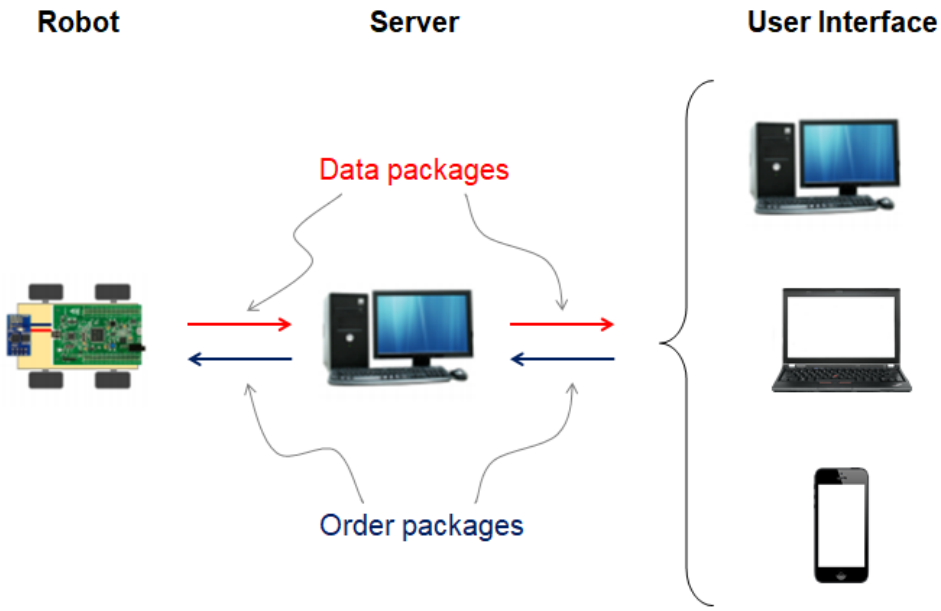

Figure 6: Scheme of the two package flows: data packages and order packages

As it is shown in Figure 6, the data packages are sent by each robot, received by the server, where they are processed and stored in a file, and finally this file is read by the user interface, which will plot the selected information.

Concurrently, the order packages are sent by the user interface and go to the server, where after being processed will be sent to the corresponding robots.

Next, the structure of both data and order packages will be explained.

4.1.1 Data packages

The data packages, which are sent by each robot and processed by the server, contain information about the robots local variables. There are two kinds of them: the ones that are sent at the beginning of the programme indicating which variables will be sent, which will be named *initialisation packages*; and the ones containing the values of those variables, which will be named *state packages*. Here are they explained:

- **Initialisation packages**: these packages tell the server how to interpret the following state packages. This information is sent during the initialisation process of each robot and is necessary due to the fact that the different robots might send their state packages in different formats. Concretely, the initialisation packages indicate in which position of the packages each variable is sent.

The initialisation packages comprise a set of bytes of type *char*, the first of which is an 'I', which acts as a marker and identifies it as an initialisation package. The rest of its characters are the names of the variables that will be sent in the state packages. Once the programme in the server has received this initialisation package, it can receive the state packages and interpret them.

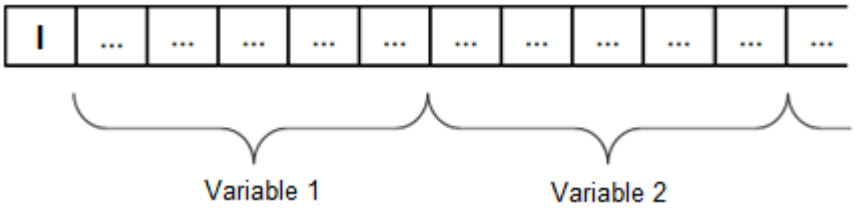

Figure 7: Structure of an initialisation package

As shown in Figure 7, the first byte is the marker 'I' and the rest are split in groups of 5. The reason why each variable name must be at most 5 characters long is that it is preferable that the packages are not too long, whereas it has been considered that 5 characters are enough to describe a variable.

- **State packages**: once the initialisation packages are sent and the robot whole initialisation process is completed, the numeric values of the variables are sent regularly every T_s in the state packages. In order to ease the server decoding process, it has been agreed that all the variables must be of type *float* and therefore must have a size of 4 bytes. This way, excluding the first byte of the packages, which is the marker 'E', the rest of the bytes will be split in groups of 4 and decoded as floats (Figure 8).

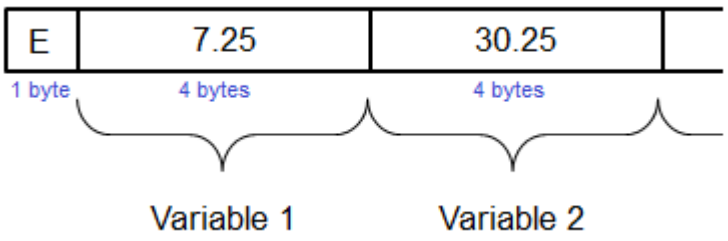

Figure 8: Structure of a state package

As seen above, the data packages are sent by the robots and can be either initialisation packages, which tell the system how to decode the following packages, or state packages, which contain the values of the robots local variables, whose information will be decoded and sent to the user interface so as to be plotted.

4.1.2 Order packages

The order packages are the packages that are sent to a robot or a group of robots, and they follow the opposite direction to the data packages: they are generated at the user interface and sent to the corresponding robots after being processed by the server. The orders can include to start, to stop, to change lane with a specific angle or to change one of its parameters, such as their current speed or their maximum speed.

The format of the order packages is a string whose first character is '>'. The table 1 contains some examples of order packages:

Order	**Package**
start	">S"
stop	">P"
change to the left lane with an angle of 35°	">C+035"
change to the right lane with an angle of 20°	">C-020"
set current speed to 0.4 rad/s	">U1=0.4"

Table 1: Example of order packages

How the user interface tells the server to which robots the orders must be sent will be explained in Section 4.2.

4.1.3 User interface

The user interface comprises a set of webpages which any device that is connected to the same wireless network can access. It has two main functionalities:

1. Plotting the information from the state packages received from the robots
2. Generating order packages that will be sent to the corresponding robots

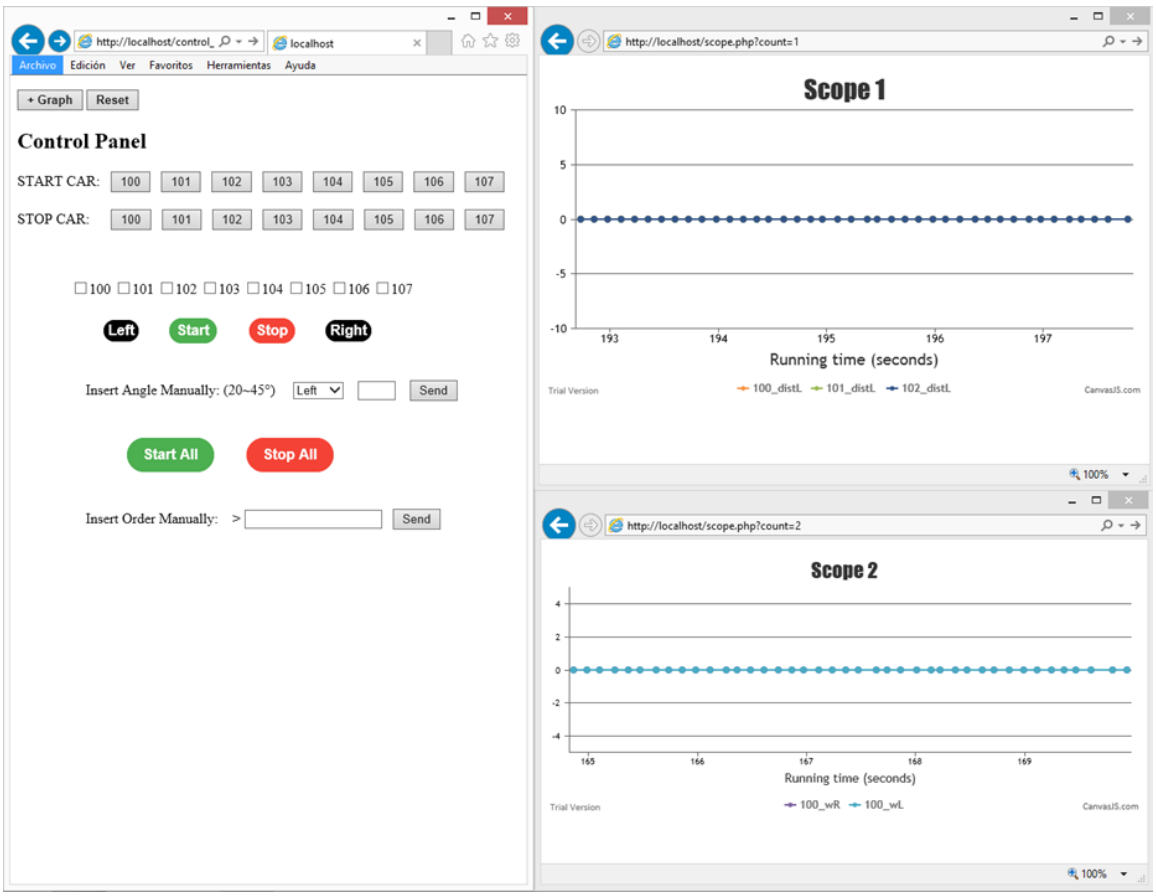

Figure 9: Screenshot of the user interface. On the left, there is the control panel; and on the right, there are two scope windows.

As it can be observed in Figure 9, the user can control the different robots by sending orders with the control panel (on the left) while observing, on the scope, the values of the variables that were previously selected (on the right).

This will be explained in detail in Section 5.

4.2 Development of the communications system

This section aims to explain in detail how the the server processes the data received from the robots and stores it, how the user interface extracts this information and plots it and how the orders are passed from the user interface to the server to be finally sent to the corresponding robot; that is a close description of the processes explained in the previous section.

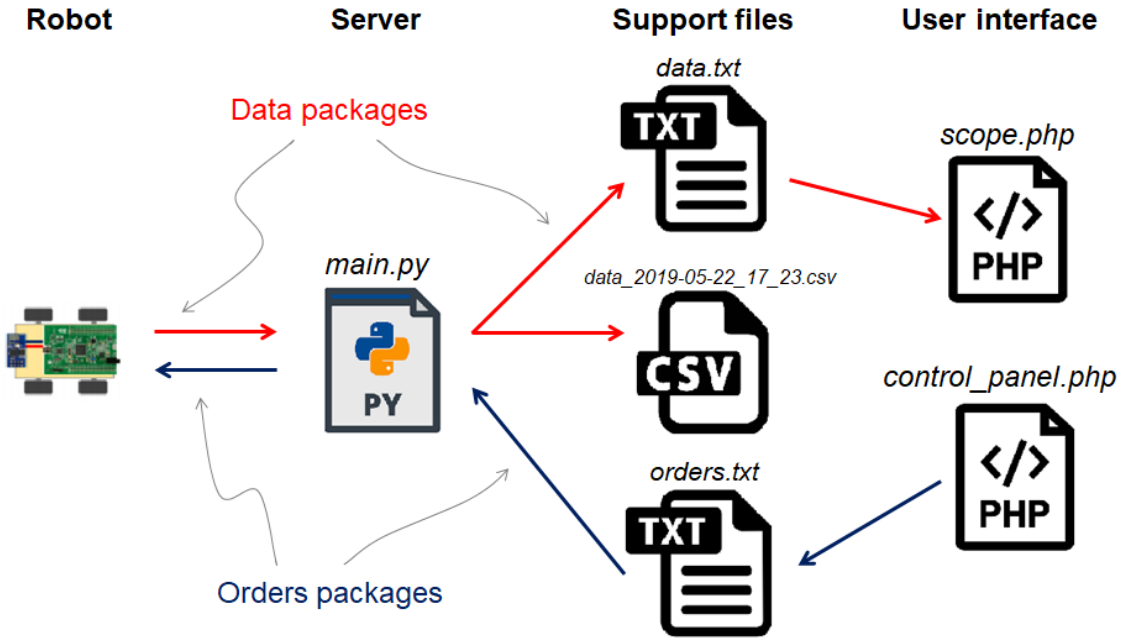

Figure 10: Detailed scheme of the communication system and the files involved, with their package flows

As it is shown in Figure 10, which is an extension of Figure 6, there are two flows of information: the data packages flow and the order packages flow. Next will all these processes be explained.

4.2.1 Handling of data packages flow

When an initialisation package is received by the server, it is processed to tell the programme how the following state packages are configured.

As previously explained, the initialisation package is split in groups of 5 characters to assign a name to each variable; this name will contribute to fill the *variables_order* Python dictionary, which contains the configurations of all the robots.

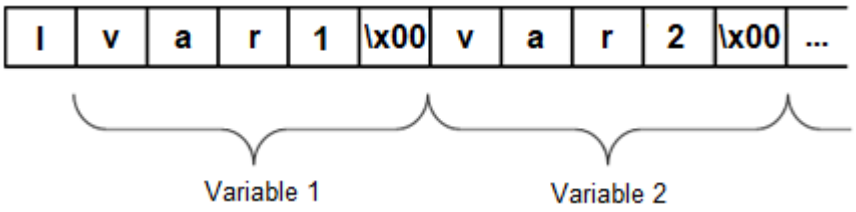

Figure 11: Format of an initialisation package

In Figure 11 the format of an initialisation package is shown. The *\x00* character stands for "void character" and will be stripped out when interpreting the variables name.

This process is done by the *main.py* programme (Appendix A), which then fills the *variables_order* dictionary as shown in Figure 12.

```
variables_order = {
        'robot_id': {'1': 'var1', '2': 'var2', ...},
        ...
}
```

Figure 12: The *variables_order* Python dictionary, which stores the information taken from the initialisation package

After having received a robot initialisation package, the programme will start to receive state packages like the previously shown example, Figure 8.

As explained before, the state packages are split in groups of 4 so as to decrypt them as floats. Thanks to the *variables_order* dictionary, the programme will be able to relate the position of each bytes group in the package with the name of the variable to which they are related. Then, the name and the value are stored in another Python dictionary: the *data* dictionary, as shown in Figure 13, following the values of the previous example.

```
data = {
        'robot_id': {'var1': 7.25, 'var2': 30.25, ...},
        ...
}
```

Figure 13: The *data* Python dictionary, which uses the *variables_order* information and stores the values corresponding to the different variables

As state packages are being received by the *main.py* programme, the data dictionary gets to change its values; once every T_s, the full data dictionary is written into two auxiliary documents: *data.txt*, which only contains the current value of each variable and that will be used by the user interface, and *data*_[timestamp].*csv*, which contains the data of the whole experiment for a posterior analysis.

Subsequently, the user interface proceeds to extract the data, filter it and plot it. This is done by opening *data.txt* and importing the dictionary in a JSON format, then picking only those variables that have been previously selected by the user on the user interface and plotting its values in a dynamic graph.

The three webpages that constitute the user interface are *control_panel* (Appendix B.I), *form.php* (Appendix B.II) and *scope.php* (Appendix B.III), and their features will be explained in detail in Section 5.

4.2.2 Handling of order packages flow

When the user selects an order, the user interface opens a TXT document called *orders.txt* and writes the order and a code referring the corresponding robots ID's on it.

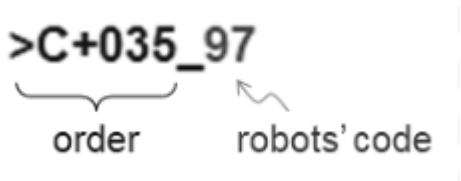

Figure 14: Example of an order and its robots code

In this example (Figure 14), the order is to change lane (>C) to the left (+) with an angle of 35° (035), and it is sent to the robots whose ID's are 100, 105 and 106. This is calculated the following way: $\sum_{i=1}^{n} 2^{\mathbf{robot\_id_i}-100} = 2^{\mathbf{100}-100} + 2^{\mathbf{105}-100} + 2^{\mathbf{106}-100} = 1 + 32 + 64 = 97$.

Then, the *main.py* programme reads the *orders.txt* file and sends the order to the corresponding robots.

5 User guide

This section contains a user guide to run the communications system.

First of all, make sure that the PC that is going to be the server has Apache24 and PHP installed. Otherwise, they must be downloaded and installed from `https://httpd.apache.org/download.cgi` and `https://www.php.net/downloads.php`, respectively.

The second thing to check is that the server and all the other devices that will access the user interface are connected to the same wireless network.

And finally, the server firewall must be disabled so the other devices can access it.

Once the previous conditions are checked, follow these steps in this order:

1. Run the file *C:\Apache24\bin\ApacheMonitor.exe*. This will set up a server thanks to which other devices can access the user interface.

2. Switch on the development board of the desired robots and wait 5 seconds before executing step 3.

3. Run the *main.py* script on the server.

4. Switch on the robot engines.

   After 15 more seconds, the robot will start sending packages, which is noticeable because it will be turning an orange led on and off intermittently at a high frequency. If after this time the server hasn't been able to receive them, a timeout exception will raise. In that case, switch off the engines and the development board and go back to step 2. Otherwise, proceed with step 5.

5. Open your web browser and enter this address: `http://localhost/control_panel.php`.

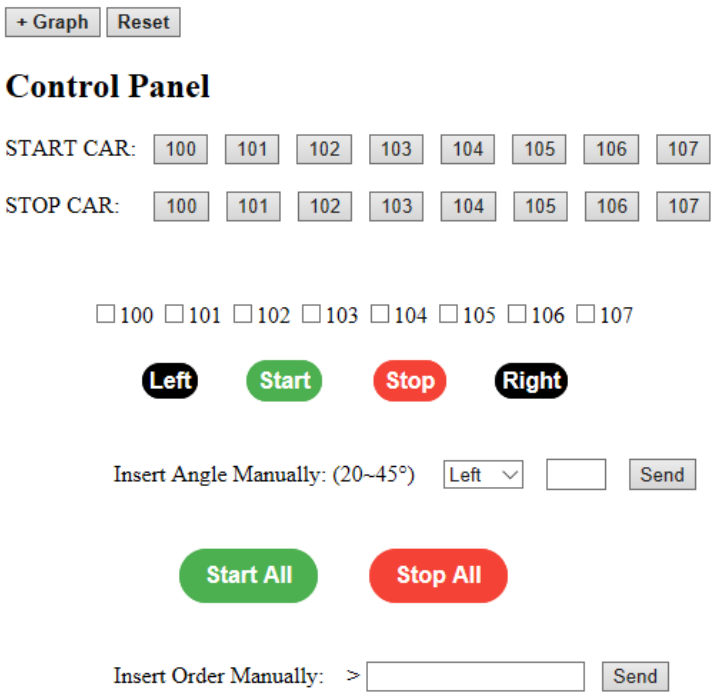

Figure 15: Screenshot of the user interface control panel

The Figure 15 is the user interface control panel. To send an order, the user can either press the grey buttons for an individual and fast action, press the Start All / Stop All buttons to refer to all the robots, use the multiple-choice selector to select a group of robots. In the latter case, the user can choose to order a set of robots to start, stop, change lane to its left/right with the default angle of 30° (*Right/Left* black buttons) or with a particular angle; or lastly, to insert an order manually.

6. Now, press the **+ Graph** button and, in the new window that will pop up (Figure 16), select the desired variables and the corresponding robots (plus optional settings) and press *Submit*.

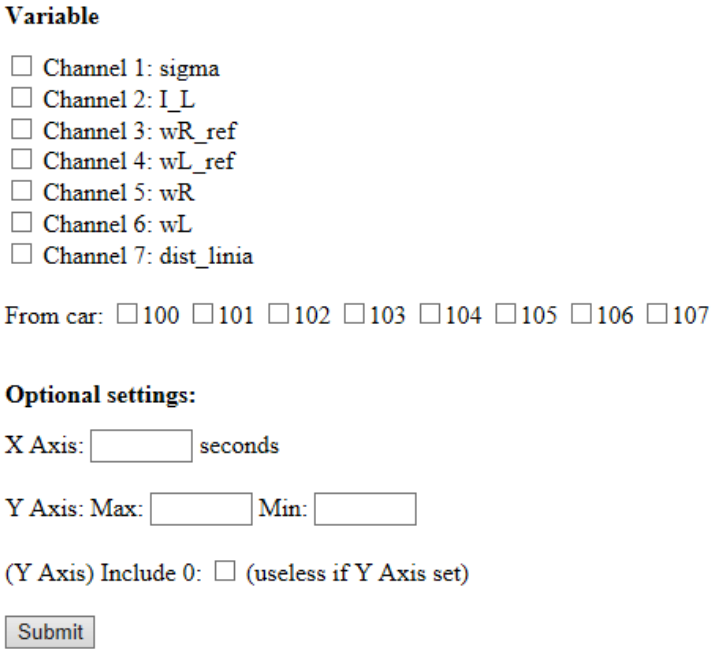

Figure 16: Screenshot of the user interface HTML form that leads to the scope

7. After submitting the previous HTML form, the same window will turn into a scope with the desired variables plotted on it. The scope looks like Figure 17.

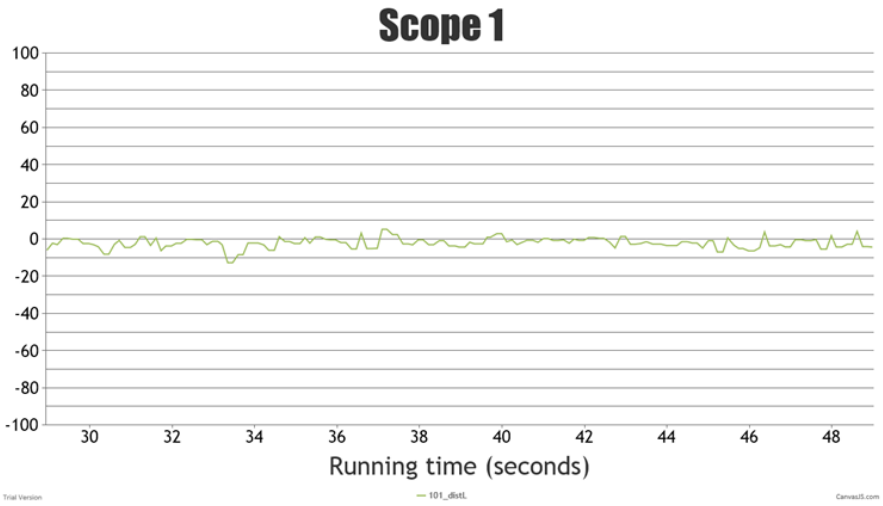

Figure 17: Screenshot of the user interface scope

Here some of the details and features of the scope webpage:

- The title is “Scope” + the number of scopes that have been created since the beginning of the experiment. To reset the counter, the user must press the Reset button on the control panel.

- The variables that are being plotted can be easily hidden by clicking on them. Thanks to that, the user may select a big range of variables and robots and go selecting and deselecting the variables that are to be scoped.

- The *CanvasJS.com / Trial version* watermark doesn’t affect the performance. The page will not expire or stop working.

- The name of the variables in the scope are [robot name]_[variable].

- There is no limit of scope windows to be created, and the scopes are sized according to the window size, which adds flexibility to the user’s experience.

It is advisable to set the screen as shown in Figure 9. On one side, the user can control the robots actions, and on the other one, the user can observe the received information.

6 New functionalities in the line tracking system

The current line tracking system is fundamentally based on a controller that receives an input from a set of light sensors and calculates a response for the wheel engines [2]. It is not among the goals of this project to improve its performance by modifying this controller parameters or by replacing it, as another student is currently doing [3], but it is to include new functionalities that could offer new trajectories or that could make the line tracking more robust.

The two functionalities that have been added to the tracking system are the *lane change* and the *recovery from loss* functions. In order to do that, the *main.c* (Appendix C.I) and the *comunicacions.c* (Appendix C.II) scripts were modified.

6.1 The lane change function

The lane change functionality is activated when the robot receives an order package whose first byte is the 'C' character. Once the lane change function is activated, the line tracking function is instantaneously disabled and the robot starts to spin at a constant velocity in the indicated direction. In order to achieve the desired angle θ, as referred in Figure 18, the robot must calculate the duration of the spin T.

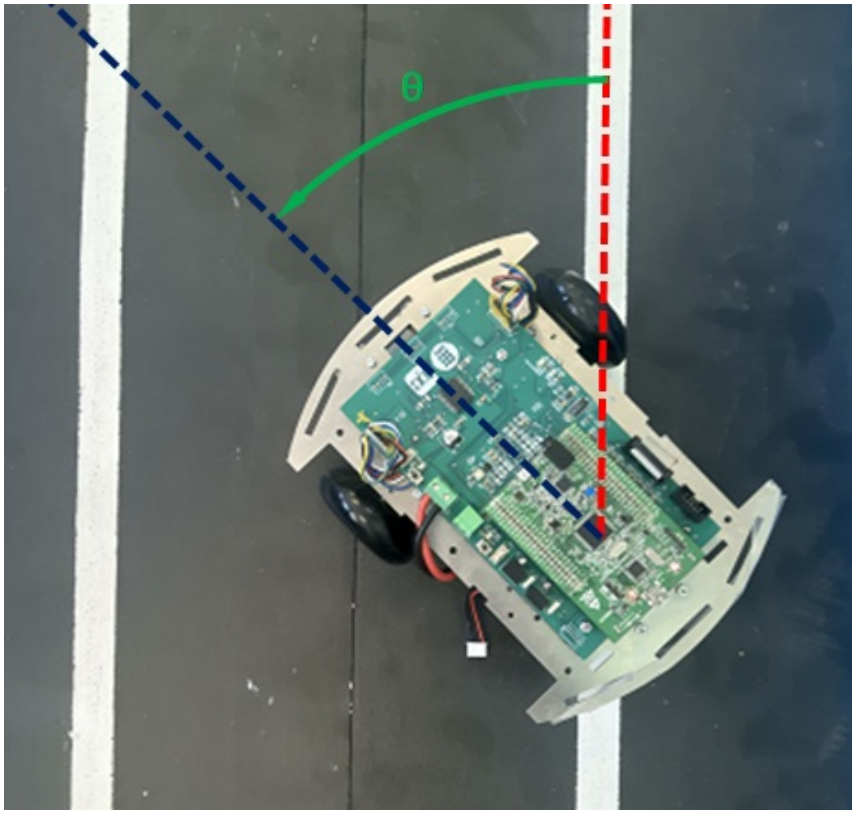

Figure 18: Picture from above of a robot in its initial stage of a lane change

We can do so analytically by describing the instant angle θ:

$$d\theta = \omega(t)\ dt \tag{1}$$

where t stands for time and ω is the spin angular velocity. By integrating with respect to time t, that is:

$$\theta = \int_{t}^{t+T} \omega(t)\ dt \tag{2}$$

being T the time of spin. As $\omega(t)$ is constant, i.e. $\omega(t) = \omega$, it can be pulled out of the integral:

$$\theta = \int_{t}^{t+T} \omega\ dt = \omega \int_{t}^{t+T} dt = \omega\ T + C(t) \tag{3}$$

where $C(t)$ is a constant that depends on the angle condition at time t. Assuming that the lane change manoeuvre occurs when the robot is aligned with the tracking line, $C(t_\mathrm{i}) = 0$.

So, now the total time of spin T is calculated:

$$T = \theta/\omega \tag{4}$$

In the code, in stead of calculating the total time and keeping track of it, the current angle is calculated by integration: every iteration of the programme, the function *update_angles* adds $\omega \cdot \Delta t$ to the current angle measure, being Δt the time difference between the last iteration and the current one and ω the current spin angular velocity (Figure 19).

```
int ticks = HAL_GetTick();
float delta_time = (ticks - *ticks0)/1000.;
*ticks0 = ticks;

*current_angle_rad_abs = *current_angle_rad_abs + delta_time*u2;
*current_angle_rad_rel = *current_angle_rad_abs - current_angle_rad_abs_0;
```

Figure 19: Fragment of the *update_angles* function. First, the time difference since the initialisation of the SysTick timer is requested; then, the time difference between it and the last iteration time is calculated; and finally, the angle between the tracking line and the current orientation, *current_angle_rad_rel*, is calculated by multiplying this time difference by the spin angular velocity, *u2*.

Once the current angle measure equals or surpasses the desired angle value, the spin is off and the robot moves straight until it finds a new white line, when the tracking mode will be instantly restored.

This new functionality remarked the importance of an old problem: the lack of memory in case of loss. For example, if in the final stage of the lane change, the robot crosses another line almost perpendicularly, the controller will not be able to recover the line tracking and will go by it, which will result in the loss of the tracking. This phenomenon also happened in the usual tracking system when the robot passed by unfortunate curves or joints of the wooden pieces. In order to solve this, the recovery from loss function, explained in the following section, was added.

6.2 The recovery from loss function

This function is applied when the robot sensors can't detect any white (the line colour). As a result, the robot goes back to the line as if it knew where it was. In practise, the performance is spectacular: it is now almost impossible for a robot to lose track of a line and get lost: even when the robot crosses a line almost perpendicularly, it is able to turn to its closest side to the line and keep tracking it.

The problem of getting lost when the robot couldn't see the line anymore was related to how the tracking system worked: the distance between the sensors plate and the line was calculated with the difference among the values of the light sensors (Figure 20); according to this distance to the line, the controller would regulate the current that would go to the engines. When a robot lost the line tracking, it was completely disoriented because it could not see anything that indicated it where to turn to recover the line.

$$distance\_measured = -\,[\,c_0(sensor_0 - sensor_7) \;+\; c_1(sensor_1 - sensor_6) \;+\; c_2(sensor_2 - sensor_5) \;+\; c_3(sensor_3 - sensor_4)\,]$$

Figure 20: How the distance to the line is measured by the robot, being c_i experimental constants and $sensor_i$ the output of the optic sensors, which are named 0 to 7, arranged from right to left

The solution to that problem was found in the process of updating the distance to the line. This distance is calculated the same way as before, but now it is only updated if at least one of the light sensors detects a white line. This way, if a robot completely crosses a line and is fully on black, it will act as if the line were still being detected by the last sensor that actually detected it previously. This process is described in the following pictures.

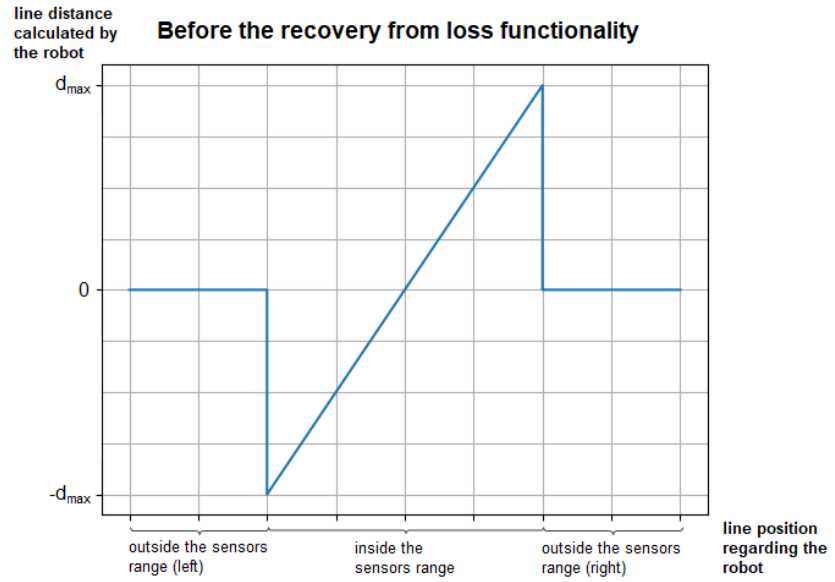

Figure 21: Representation of the robot behaviour regarding the calculation of its distance to the tracking line, before the implementation of the recovery from loss functionality

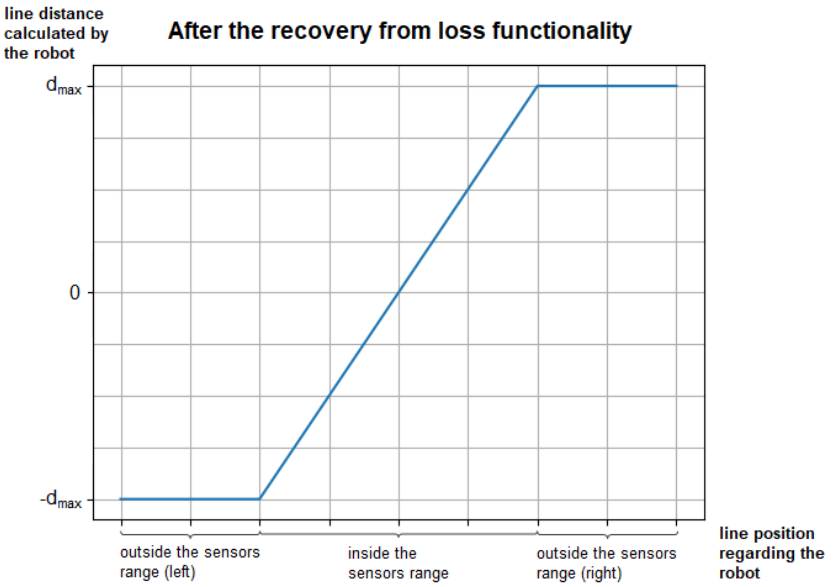

Figure 22: Representation of the robot behaviour regarding the calculation of its distance to the tracking line, after the implementation of the recovery from loss functionality

Before the implementation of the recovery from loss functionality (Figure 21), the robot was unable to know what side it had to turn to recover the lost line. This can be noted because the distance to the line value is 0 when the robot can't see the line and is fully on black (outside the sensors range).

Contrarily, after its implementation (Figure 22), the robot is able to "remember" where the line was when it was detected for the last time as if the line were being detected at its saturation distance d_{max}, which is the maximum distance that can be calculated with the sensors outputs. Therefore, if the sequence of Figure 23 were reproduced, the robot variable "distance to the line" would be of the value $+d_{\text{max}}$ and it would then redirect itself back to it.

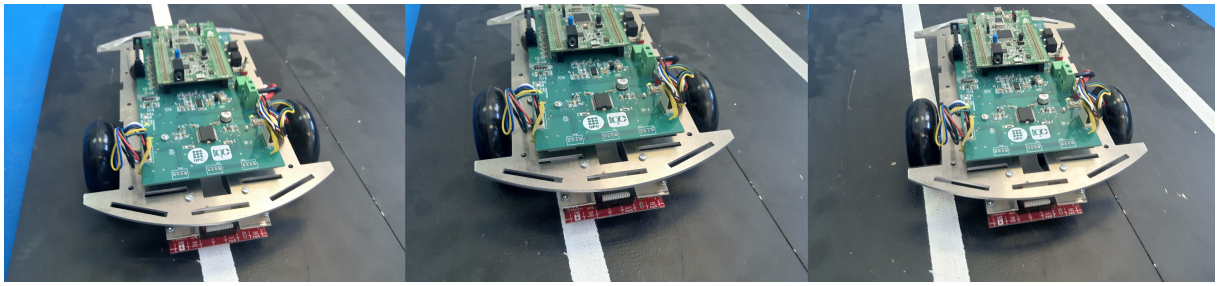

Figure 23: Chronological sequence of pictures showing a robot being moved to its left until its sensors can't detect the line

6.3 Examples of lane changes and their analysis

6.3.1 Example 1. Lane change of 60° to the left

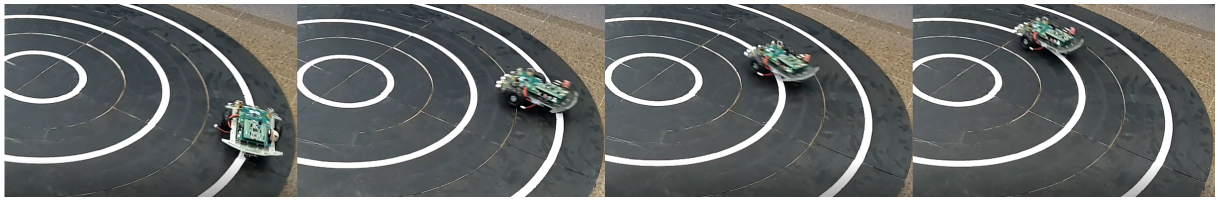

Figure 24: Chronological sequence of pictures showing a robot making a lane change to its left with an angle of 60°

This is an example of a lane change of 60° to the left and here are all the stages of the sequence (Figure 24) detailed:

1. The robot is tracking the outer line when suddenly receives an order to make a lane change of 60° to its left.

2. The robot turns 60° to its left and then moves forward looking for a new white line.

3. The robot finds the new line and gets to track it without any problems.

4. The controller reaction is stable and the robot resumes the tracking.

And here is the analysis of their distance to the line, with all the described instants marked:

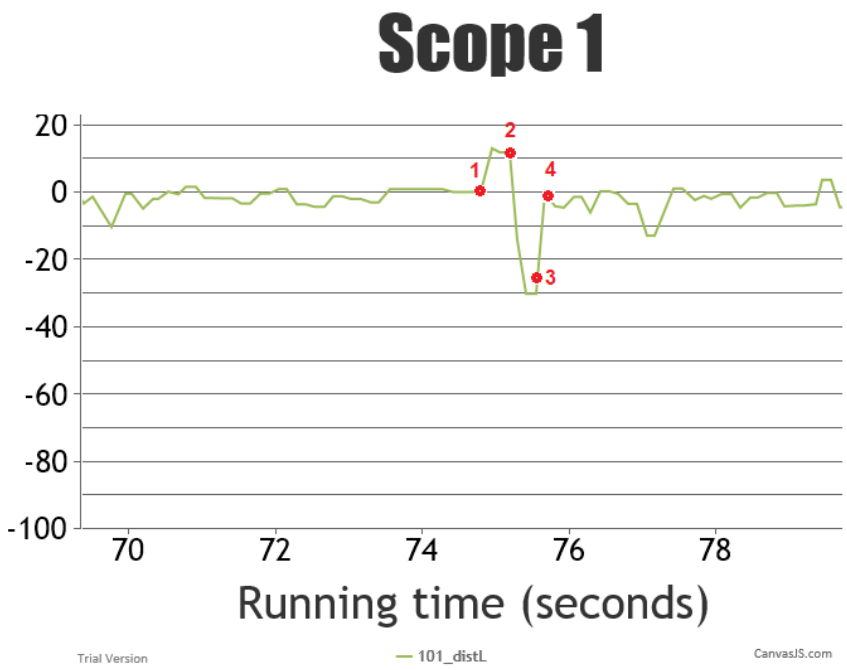

Figure 25: Analysis of the distance to the line during the previous lane change by marking the above described instants on the scope

As it is shown in Figure 25, the value of the distance to the line variable is very close to 0 before the lane change starts (Instant 1); that is because the curvature of the outer circumference is small and the robot can track the line smoothly.

After leaving the old line and on the point of arriving to the new line (Instant 2), the distance to the line variable is set at its positive saturation distance (about +12mm) because the last sensor that detected the old line is the one on the top right, which concludes that the line is on the right side (positive distance). Then the sensors on the left detect the new line and conclude that the line is on the left (negative distance) at Instant 3.

When the line tracking is normalised again (Instant 4), the average distance is 0, but now it will be less smooth due to the bigger curvature of the new line, making higher peaks.

6.3.2 Example 2. Lane change of 90° to the left

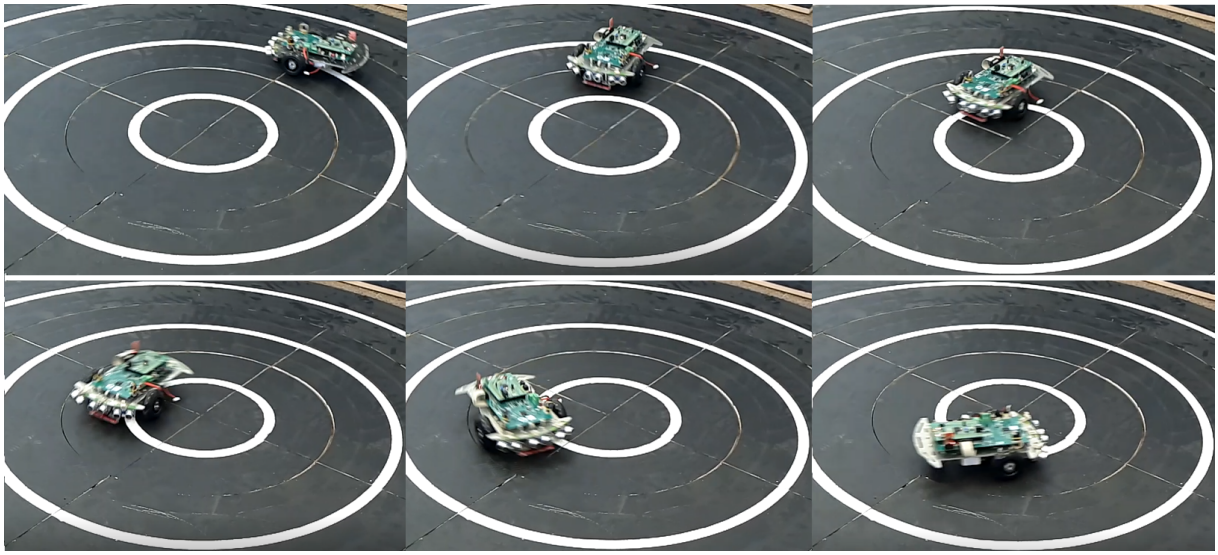

Figure 26: Chronological sequence of pictures showing a robot making a lane change to its left with an angle of 90°

This is an example of a lane change of 90° to the left and here are all the stages of the sequence (Figure 26) detailed:

1. The robot is tracking the middle line when suddenly receives an order to make a lane change of 90° to its left.

2. The robot turns 90° to its left and then moves forward looking for a new white line.

3. The robot finds the new line and gets to track it, apparently without problem.

4. The response of the controller is too aggressive and the robot loses the track of the line. Without the memory functionality, the robot would now be lost.

5. The robot "remembers" that the line is at its left because it has not updated the distance to line variable since the last time a sensor detected some white (which was the most at left sensor, since the robot was turning right). Actually, the controller will act as if the line were being detected by that sensor. In this case, it turns left to recover the line, but at this instant it has not found it yet.

6. The robot finally finds the line that had previously lost and goes on with the tracking.

And here is the analysis of their distance to the line, with all the described instants marked:

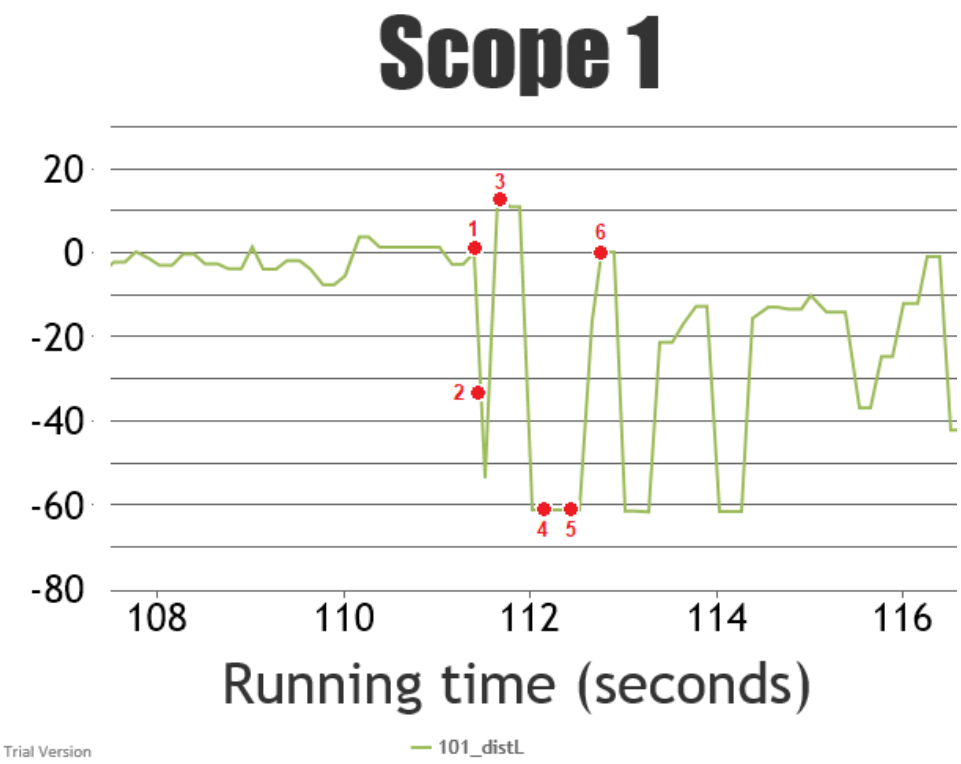

Figure 27: Analysis of the distance to the line during the previous lane change by marking the above described instants on the scope

In this example (Figure 27) the difference between the behaviour of the robot before and after the lane change is more noticeable due to the abrupt increase of the line curvature. The inner line radius of curvature is so small that the robot makes sudden changes left and right to track the line (after Instant 6). The average then is not 0 but negative, which means that the robot is constantly getting off the line on its right side (negative peaks) and redirecting itself back to the center (distance = 0).

The behaviour of the robot during the lane change starts like the previous example: the distance to the line variable is around 0 until the lane change starts (Instant 1), when it changes to its positive saturation distance (probably this package was overwritten by a posterior package or was delayed due to the low priority of the communications system in the robots; the value should be around 12mm). The distance to the line value is kept at Instant 2 because no line is detected, even though the scope makes a decreasing line between data points. When the robot detects the new line on its left (between Instants 2 and 3), the distance to the line variable is negative, and goes increasing as the robot goes by the line. At Instant 3, the robot has come to the other side of the line and the distance to the line is positive.

However, at Instant 4 the robot gets lost on the right side of the line (negative saturation distance, about -60mm), and turns left to recover it. At Instant 5, no line has been found yet, so the distance to the line variable still equals the negative saturation distance and the robot keeps turning left. At Instant 6, the robot has recovered the line tracking.

6.3.3 Example 3. Lane change of 30° to the right

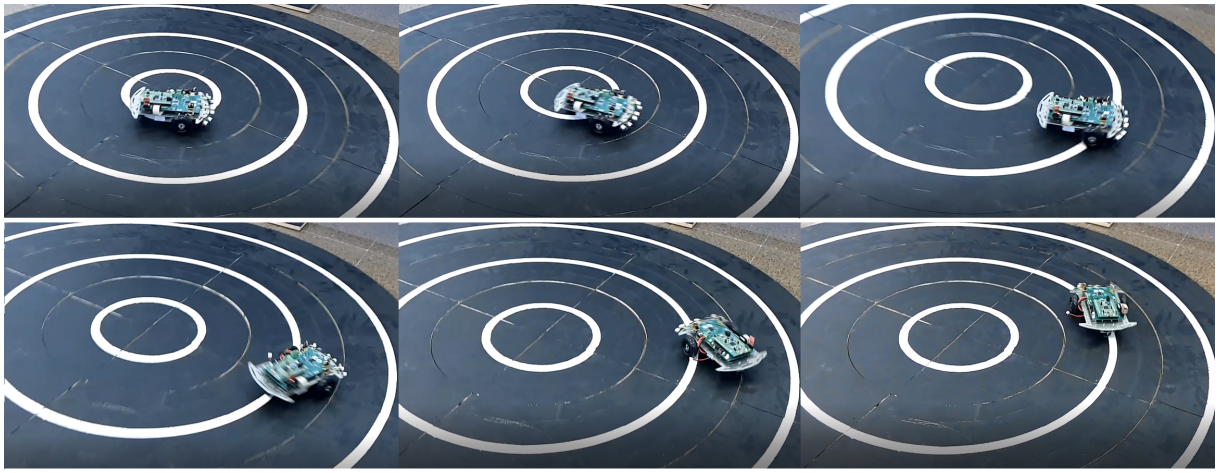

Figure 28: Chronological sequence of pictures showing a robot making a lane change to its right with an angle of 30°

This is an example of a lane change of 30° to the right and here are all the stages of the sequence (Figure 28) detailed:

1. The robot is tracking the inner line when suddenly receives an order to make a lane change of 30° to its right.

2. The robot turns 30° to its right and then moves forward looking for a new white line.

3. The robot crosses the new line almost perpendicularly, and its reaction is too weak to track the new line.

4. Even though the robot went past the line, it registered its distance to the line when one of its optic sensors detected white as latest. Since this sensor is the one on the left, the robot turns left as if the line were still being detected by that sensor.

5. The robot keeps turning left until it finds the lost line.

6. The robot finally finds the line and keeps tracking it.

And here is the analysis of their distance to the line, with all the described instants marked.

Figure 29: Analysis of the distance to the line during the previous lane change by marking the above described instants on the scope

This last example is explained in Figure 29. At Instant 1, when it receives the order, the car is out (or almost) of the line and the distance to the line variable is not updated, which explains why the distance to the line value is not decreased to the negative saturation distance at Instant 2.

Just when the top right sensor detects the new line (Instant 3a) the distance to the line equals the positive saturation distance, but quickly the robot crosses the line (Instant 3b) and the value decreases to the negative saturation distance, which will remain unchanged until Instant 5, when the robot detects the line again. After crossing again the line (Instant 6), the tracking system is fully recovered.

In the last two examples, it is to be noted that when the robot is fully on black, i.e. it doesn't detect any white line, the distance to the line variable is not updated, which results in a horizontal line on the Scope (Points 4 and 5 from Figure 27 and Points 3b and 4 from Figure 29).

The fact that the positive saturation distance and the negative saturation distance have different values is an issue regarding the calibration of the sensors outputs, but it doesn't have a noticeable impact on the robots, as the recovery from lost function works perfect on both sides.

7 Budget

The total cost of this project is constituted of the costs of the robots assemblage and all the equipment used, the licence costs of the programmes involved and the remuneration for the time spent on it. Here are they described:

The costs of the robots assemblage comprise the remuneration for the time spent on their assemblage and the purchase of their materials. The approximate time of the assemblage is 4h and an approximate wage for a technician is 25€/hour, which makes it worth 100€. The price of the materials is described in Table 2 and concludes a total cost of 255.18€ per robot. The total cost of the robots assemblage is then 355.18€ per robot, plus 69.00€ for its battery charger.

Even though the system can encompass 8 robots at most, this project has been developed using only 2 robots and so will be reflected in the total costs. Then, the total cost of the 2 robots assemblage plus 1 charger is **779.36€**.

Concept	**PVP [€]**	**Units**	**Total [€]**
DC/DC Converter 12V/5V	5.49	1	5.49
DC/DC Converter 12V/3.3V	2.35	1	2.35
Motor Driver	6.49	1	6.49
Development Board	17.58	1	17.58
DC Motor	29.45	2	58.90
Gearmotor Bracket Pair	6.36	1	6.36
Scooter/Skate Wheel	2.16	2	4.32
Wheel Adapter	5.55	2	11.10
Support Wheel	6.45	1	6.45
Distance Sensor	16.74	3	50.22
Line Sensor	8.06	1	8.06
Wi-Fi Module	6.40	1	6.40
Battery	32.50	1	32.50
Connectors	1.10	1	1.10
PCB 1	21.66	1	21.66
PCB 2	16.20	1	16.20
Total costs per robot			**255.18**

Table 2: Assemblage costs per robot. The prices have been updated on June 22nd 2019.

The licence costs only include the Canvas licence, which is the only one required to make a commercial use of this project, and it is worth $399 (∼**360€**).

The other equipment that has been used is a 700€ laptop, a 500€ desktop computer and a 30€ router, which result in **1,230€**.

And finally, the labour costs constitute the remuneration for the time spent creating the scripts that comprise this project, learning the fields of interest of the different programming languages: PHP, Javascript/HTML, Python (sockets) and C; also, experimenting with the robots performance in the circuits used, and finally writing down the project. It is mainly the investigation on how to apply certain functions in the several programming languages and the trial and error on their performance and their implementation to the robots that have hogged the most of the labour time and increased it that much.

The amount of hours devoted to the project, since its start in October 15th 2018 and approximating a dedication of 3h per workday, is calculated to be of 516 hours. Being 35€/hour the corresponding engineer salary, that makes the project labour costs worth **18,060€**.

All the described parts are captured in Table 3, stating that the total cost of the project is **24,719.53€**.

Concept	**Total costs [€]**
Robots costs	779.36
Licence costs	360.00
Equipment costs	1,230.00
Labour costs	18,060.00
Subtotal	20,429.36
VAT (21%)	4,290.17
TOTAL	**24,719.53**

Table 3: Total cost of the project

8 Environmental impact

This project has consisted in the development of scripts that improved the performance of an already existing system and that added new functionalities to it. Therefore, and since no hardware or any physical device has been added to the system, the only environmental impact that may be considered during the development of this project is the electricity used by both the computer and the robots.

Conclusions

Both goals have been successfully completed: on the one hand, a functional communications system has been successfully created. It can be accessed by multiple devices at once and allows operating with several robots sending different variables. It also saves the all the data in a CSV file so data can be analysed after each experiment. In general, the new user interface allows to control and analyse the robots performance easily and accurately, as well as to work remotely by accessing the wireless network.

And on the other hand, the lane change and the recovery from loss functionalities have been added and they both make the system more robust and expand the robots trajectory options. New projects may combine the lane change functionality with the ultrasounds sensors or the communication among the different robots, so as to create a completely independent system of autonomous line tracking robots.

Acknowledgements

First, I would like to thank my tutor Arnau Dòria for his patience, his constant feedback and his advice, as well as for his perpetual willingness to meet and help.

And secondly, to Victor Repecho del Corral for helping me understand how the previous communications system and the robots script worked, as well as for continuously fixing and recharging the robots I was using.

Bibliography

[1] Antoni Riera Seguí, *Disseny i implementació d'un sistema de comunicacions WiFi per a una xarxa de vehicles autònoms*, Bachelor Final Project, Universitat Politècnica de Catalunya, 2016.

[2] Ivan Prats Martinho, *Control design and implementation for a line tracker vehicle*, Bachelor Final Project, Universitat Politècnica de Catalunya, 2016.

[3] Marc Murcia Matute, *State feedback control and Luenberger observer design for a differential-drive mobile robot*. Bachelor Final Project, Universitat Politècnica de Catalunya, 2019.

[4] ST Microelectronics, *User Manual - Discovery kit with STM32F407VG MCU*, 2017. Retrieved from https://www.st.com/content/ccc/resource/technical/document/user_manual/70/fe/4a/3f/e7/e1/4f/7d/DM00039084.pdf/files/DM00039084.pdf/jcr:content/translations/en.DM00039084.pdf.

[5] Espressif Systems, *ESP8266 Overview*, 2019. Retrieved from https://www.espressif.com/en/products/hardware/esp8266ex/overview.

[6] Espressif Systems, . *ESP8266EX Datasheet*, 2018. Retrieved from https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.

[7] GRobotronics, *ESP8266 WiFi Module*, 2019. Retrieved from https://grobotronics.com/esp8266-wifi-module.html?sl=en.

[8] Licesio J. Rodríguez-Aragón, Universidad Rey Juan Carlos. *Tema 4: Internet y Teleinformática*, 2019. Retrieved from https://previa.uclm.es/profesorado/licesio/Docencia/IB/IBTema4.pdf.

[9] Sergio de Luz, *La capa de transporte en Internet: TCP y UDP sobre IP. Volumen I*, 2011. Retrieved from https://www.redeszone.net/2011/01/28/la-capa-de-transporte-en-internet-tcp-y-udp-sobre-ip-volumen-i/.

[10] Transfer Multisort Elektronik S.L.U.. *POLOLU-960*, 2019. Retrieved from https://www.tme.eu/html/ES/sensores-reflectantes-de-linea/ramka_15422_ES_pelny.html.